IMPLEMENTATION OF AN EXACT PENALTY FUNCTION FORMULATION TO SOLVE CONVEX NONLINEAR PROGRAMMING PROBLEMS

By

Christopher W. Fowler

A Project Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

Approved:

_____

Joseph G. Ecker
Project Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 8, 1991
(For Graduation May 1991)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1991	Final

4. TITLE AND SUBTITLE
Implementation of an Exact Penalty Function Formulation to Solve Convex Nonlinear Programming Problems

5. FUNDING NUMBERS

6. AUTHOR(S)
CPT Christopher W. Fowler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Masters Project
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Mathematical Sciences
United States Military Academy
West Point, NY 10996

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
UL

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A convex nonlinear programming problem can be reformulated using an exact penalty function. One such reformulation is proposed by Zangwill in [11]. The result of this reformulation is an unconstrained problem. Unfortunately, this nonlinear, convex function is not a continuously differentiable function. Most computer based algorithms operate on the condition that the functions involved are differentiable. One exception is the Ellipsoid Algorithm, EA3, by Ecker and Kupferschmid [9]. This algorithm, though demonstrating only linear convergence, is robust in solving problems with nondifferentiable functions. This project examines whether it is better to solve the original constrained problem or the reformulated unconstrained problem, where better is defined as more accurate and faster. After showing that the reformulated problem can always be solved for a convex situation, comparisons are made using the ellipsoid algorithm of accuracy of solution and solution time for several example problems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
NONLINEAR PROGRAMMING...ELLIPSOID ALGORITHM EXACT PENALTY FUNCTIONS...OPERATIONS RESEARCH OPTIMIZATION...CONSTRAINED OR UNCONSTRAINED PROBLEMS	36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

ABSTRACT

A convex nonlinear programming problem (NLP) can be reformulated using an exact penalty function. One such function is the L1 exact penalty function examined by Zangwill [11]. The result of this reformulation is an unconstrained optimization problem. Unfortunately, this nonlinear, convex function is not a continuously differentiable function. Most computer methods/algorithms are based on the condition that the functions involved are differentiable. One exception is the Ellipsoid Algorithm, EA3, by Ecker and Kupferschmid [9]. This algorithm, while demonstrating only linear convergence, has been shown to be robust in solving problems with nondifferentiable functions. The problem reduces to the following question. Is it "better" to solve the original, constrained problem, or the reformulated unconstrained problem? Here, "better" is defined as faster and more accurate. After demonstrating that the exact penalty formulation will always solve the convex NLP, comparisons are made, using the ellipsoid algorithm, of accuracy of solution and solution time for several example problems.

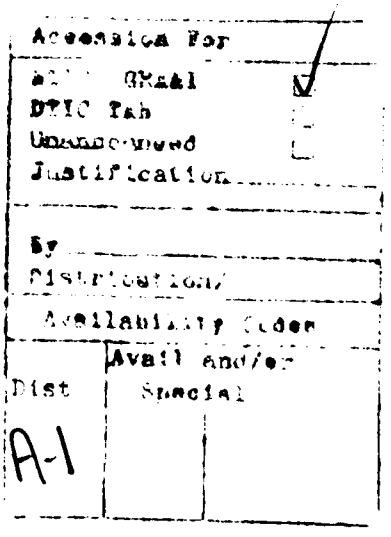


TABLE OF CONTENTS

CHAPTER 1: THE EXACT PENALTY FUNCTION

The convex nonlinear programming problem seeks to minimize an objective function, $f(\mathbf{x})$, for the n-dimensional vector $\mathbf{x}$, subject to a number of constraints, $g_i(\mathbf{x})$ for i=1,2,...,m. Written in standard form, the problem appears as:

$$minimize \quad f(x)$$

$$s.t. \quad g_i(x) \leq 0 \ , \ i=1,2,\ldots,m$$

where $f(\mathbf{x})$ and each of the $g_i(\mathbf{x})$ are convex functions. This constrained nonlinear programming problem can usually be solved by one of several algorithms. This process, however, can be difficult and time consuming. An apparently simpler problem would be to minimize an objective function, subject to no constraints. A transformation of the constrained problem can be made, resulting in an unconstrained optimization. One such reformulation is the exact penalty function presented by Zangwill in [11]. The NLP reduces to:

$$\min p(x,c) \ , \ x \in R^n$$

where the following functions and terms are defined:

$$p(x,c) = f(x) + cP(x)$$

$$f(x) = original \ objective \ function$$

$$c = penalty \ multiplier$$

$$P(x) = \sum_{i=1}^{m} \max[g_i(x), 0], \ the \ exact \ penalty \ function$$

Penalty Function Theory

The penalty function formulation introduces an increase in objective function value for infeasible points, the "penalty" added to the minimization problem. So, given a particular solution point, $\mathbf{x}_k$, if a constraint is satisfied, $g_i(\mathbf{x}_k)<=0$, then the $\max[g_i(\mathbf{x}_k),0]=0$, and the resulting penalty to the objective value is $(c)*(0)=0$. On the other hand, if the constraint is violated, $g_i(\mathbf{x}_k)>0$, then a positive component multiplied by c is added to the objective function.

The constrained problem has now been reduced to a single unconstrained minimization. However, this single function has become complicated and nondifferentiable. A method will be presented to solve this reformulated problem. First, it must be shown that this unconstrained problem can be solved, and the solution of the problem is the same solution as for the unconstrained problem. Two assumptions are required for the theorem:

1. $\mathbf{x}^*$ is the optimal point for the constrained nonlinear programming problem.

2. $S^\circ \neq \emptyset$, where S° is defined as the interior of the feasible set.

The penalty function theorem can then be stated as follows:

THEOREM: Let $\mathbf{x}^*$ be the optimal point for a convex nonlinear programming problem and the interior of the feasible set be nonempty. Then, the convex problem can be solved by a single unconstrained minimization.

The proof of this theorem can be found at appendix 1. In this

proof, the penalty multiplier is defined so that convergence to the optimal solution is guaranteed. The multiplier is defined as follows:

$$c = \frac{\beta - 1}{\alpha} \text{ , where:}$$

$$\forall\, z \in S^{\circ},\ (z \text{ is a strictly feasible point}):$$

$$\beta = f(x^*) - f(z) \qquad \alpha = \max_i [g_i(z)]$$

Since $\mathbf{x}^*$, the optimal point, is the solution being sought, an estimate for $f(\mathbf{x}^*)$ must be made in the calculation of the penalty multiplier.

Estimation of the Penalty Multiplier

The reformulated unconstrained problem is:

$$\min \quad f(x) + c \sum_{i-1}^{m} \min [g_i(x), 0]$$

However, c cannot be calculated directly. The following method can be used to make an estimate of the penalty multiplier:

<u>step 1:</u> find a $\mathbf{z}$, s.t. $g_i(\mathbf{z}) < 0$, ie: $\mathbf{z}$ is strictly feasible.

<u>step 2:</u> calculate $\alpha = \max_i [g_i(\mathbf{z})]$

<u>step 3:</u> estimate β with:

$\beta' = f' - f(z)$, *where* f' *= a lower bound on the true objective value*

<u>step 4:</u> the estimate for c becomes:

$$\bar{c} = \frac{\beta' - 1}{\alpha}$$

Now, any lower bound on the true objective value will yield a $\bar{c}$ larger than the true value of c. The proof of the penalty function theorem in appendix 1 demonstrates that any value larger than the true value of c will also cause the penalty function formulation to converge to the optimal point. Therefore, the estimate for c as calculated above, will be sufficiently large to cause the problem to converge to a solution.

An Illustrative Example

The exact penalty function formulation and solution will be illustrated by the following example

$$\min\ (x_1-4)^2+(x_2-4)^2$$

$$s.t.\quad x_1-2\le 0$$

$$x_2-2\le 0$$

$$-x_1,-x_2\le 0$$

Graphically, this example can be depicted as shown in figure 1 on the next page. The solution to this simple convex problem can be taken right from the graph or solved manually using the KKT method described in [6]. Obviously, the optimal solution to the minimization of the radius of the circle centered at (4,4) is $\mathbf{x}^*=(2,2)$ with optimal value of $f(\mathbf{x}^*)=8$. Solving for the Lagrange multipliers yields the vector: $\lambda^T=[4\ 4\ 0\ 0]$

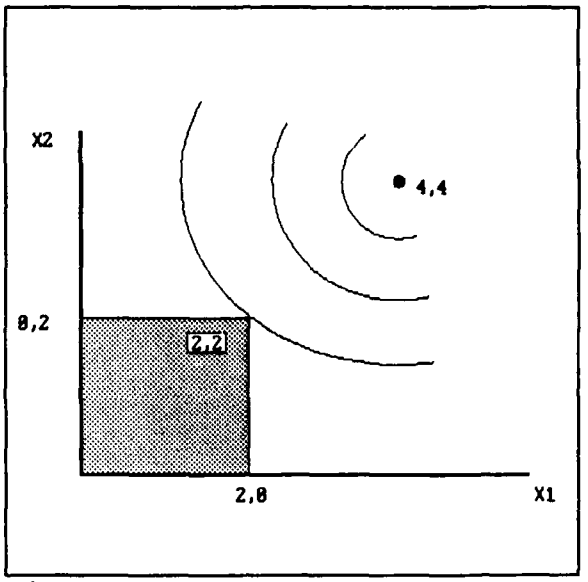


Figure 1

To solve this problem using the exact penalty function formulation, the problem will be rewritten as an unconstrained minimization:

$$\min\ (x_1-4)^2+(x_2-4)^2+\bar{c}\sum_{i-1}^{4}\max[g_i(x),0]$$

and substituting each of the four constraints into the penalty function. The remaining problem is to find a value for the penalty multiplier. To begin, select the strictly feasible point, $\mathbf{z}=(1,1)$. Calculations yield $f(\mathbf{z})=18$ and $\alpha=-1$.

Substituting all of the intermediate terms into the penalty multiplier equation yields:

$$\bar{c}=\frac{f'-f(z)-1}{\alpha}=\frac{f'-18-1}{-1}=19-f'$$

To complete the calculation of the penalty multiplier, a lower bound is needed on the true objective value.

Duality to Solve for the Penalty Multiplier

One method of obtaining a lower bound on the true optimal value is to find a dual feasible point. Using the gradient method in [1] to solve the Lagrangian Dual Problem will yield a lower bound for the primal problem being solved. For the example problem, the Lagrangian function is:

$$L(x,u) = f(x) + \sum_{i=1}^{4} u_i g_i(x)$$

Then, set up the dual problem:

$$\max \ \theta(u), \quad u \geq 0$$

where:

$$\theta(u) = \textit{infimum}\ L(x,u), \quad X \in \Re^n$$

Fortunately, all that is needed for the primal problem is any lower bound on the optimal value. Therefore, the dual can be initialized with $\mathbf{u}=(0,0)$, thus reducing $L(\mathbf{x},\mathbf{u})=f(\mathbf{x})$. So, to get a lower bound for the convex primal problem, one needs only to minimize the original objective function without any of the constraints.

For the example problem, initializing $\mathbf{u}=(0,0,0,0)$, will result in the problem: $\min f(x) = \min (x_1-4)^2+(x_2-4)^2$

which has the solution $\hat{x}=(4,4)$ *with* $f(\hat{x})=f'=0$

A value for the penalty multiplier can now be established:

$$\overline{c}=19-f'=19$$

with the final formulation for the new unconstrained minimization:

$$\min\ (x_1-4)^2+(x_2-4)^2+19\ \sum_{i-1}^{4} \max[g_i(x),0]$$

NOTE: Knowing that the optimal value for this problem is 8 allows calculation of a smaller multiplier equal to 9. In the next chapter, both of these values will be used to solve the problem.

CHAPTER 2: METHODOLOGY

In [5], [6], and [9], Ecker and Kupferschmid develop and implement an ellipsoid algorithm for solving nonlinear programming problems. For a given n-dimensional NLP, the ellipsoid algorithm generates a sequence of shrinking ellipsoids. Each of these ellipsoids contains the optimal point. The volume of the ellipsoids continues to reduce, thus the algorithm converges to the optimal solution. The version of the algorithm used in these computations was EA3, which can be found in [9].

The Ellipsoid Algorithm

The ellipsoid algorithm starts with an initial center point and initial ellipsoid:

$$E_0 = [x \in \Re^n \mid (x-x^0)^T Q_0^{-1} (x-x^0) \leq 1]$$

where $\mathbf{x^0}$ is the center point of the initial ellipsoid and Q_0 is a symmetric positive definite matrix. This initial ellipsoid contains the optimal solution (guaranteed if this ellipsoid contains the entire feasible set). The algorithm generates sequentially smaller ellipsoids through what is termed either a phase 1 or phase 2 iteration process.

A phase 1 iteration occurs if the center point of the current ellipsoid is infeasible. A cutting hyperplane calculated from the gradient of the first violated constraint is used to generate the next ellipsoid. A phase 2 iteration occurs if the center point of the current ellipsoid is feasible (there are no violated constraints). For this iteration, the cutting hyperplane is calculated using the gradient of the objective function. The general form for the

equation of the cutting hyperplane (where k is the index for the current iterate) is:

$$H_k = [x \mid -\nabla f_i(x^k)^T (x - x^k) - 0]$$

where: i= index of violated constraint for a phase 1 iterate
m+1 (objective fcn gradient) for a phase 2 iterate

In both cases, the current ellipsoid is cut in half and a new ellipsoid is generated which includes the half of the previous ellipsoid containing the optimal point. The volume reduction on each iteration is dependent on the number of variables and can be expressed as the ration of volumes of two successive ellipsoids:

$$q_n = \frac{Vol(E_{k+1})}{Vol(E_k)} = \sqrt{\frac{n-1}{n+1}} \left[\frac{n}{(n^2-1)^{1/2}} \right]^n$$

Some example values for the amount of volume reduction on each iterate are:

n= # of variables	q_n= volume reduction
2	.77
10	.9511
100	.9950

As the number of variables in the problem increases, the volume reduction for each iterate decreases. In general, the fewer the variable that exist, the faster the convergence is to the optimal solution. In its current form, the ellipsoid algorithm, EA3, will continue to run until either the optimal point is found or the current matrix, Q_k is no longer numerically positive definite.

Function and Gradient Routines

Specific routines exist for coding nonlinear programming problems to be solved by a specific algorithm. For solving general constrained problems, the NLP is coded into two fortran routines, FCN (for function calls) and GRAD (for gradient calls). These two routines are then passed as parameters to whichever algorithm/program is being used. For the example problem started in chapter 1, the function (FCN) and gradient (GRAD) routines can be found in appendix 3. Each of the problems examined were likewise coded.

Some additional subroutines are required to solve the exact penalty function formulated NLP. By it's construction the reformulated problem has no constraints, yet the constraint components are included in the objective function. Also, the ellipsoid algorithm searches for a violated constraint function before turning to the objective function. In the case of the reformulated problem, the algorithm need only look directly to the gradient of the objective function (since there are no constraints to violate, hence no gradient evaluation for a violated constraint).

Two additional subroutines FCNX and GRADX were written to take advantage of the existing FCN and GRAD routines, yet perform only the calculations needed for the exact penalty function formulation. FCNX prompts for the original number of constraints and the penalty multiplier to be used. The routine then uses the FCN routine for the specific problem to construct the complete unconstrained exact penalty function. GRADX uses the gradient information provided in the problem GRAD routine and constructs the gradient for the unconstrained objective function. The listings of FCNX and GRADX can be found at appendix 4.

Error Analysis

Error versus effort curves are generated for each of the various problems to trace the accuracy of a solution against time. The horizontal axis is the effort axis. Effort is measured by problem state central processing unit (PSCPU) time. The only time counted in solving the problem is during the steps of the actual algorithm execution. For the penalty function problem, since no constraints exist which could be violated, then there is no need to calculate function values during iteration. The gradient of the objective function is required on every iterate so only those calculations are counted for time.

The vertical axis of the error versus effort plot is titled "Log Relative Combined Solution Error." The combined error for each iterate, $\mathbf{x}^k$, is calculated as follows:

$$e(x^k) = | f_0(x^k) - f_0(x^*) | + \sum_{i=1}^{m} \lambda_i^* | f_i(x^k) |$$

where $\mathbf{x}^*$ is the optimal point and the λ_i^* are the Lagrange multipliers at optimality. These error terms are then normalized to obtain relative combined error terms:

$$E(x^k) = \frac{e(x^k)}{e(x^0)}$$

The logarithms of these $E(\mathbf{x}^k)$ error terms are then plotted against the PSCPU times. The error versus effort curves for the example problem is at appendix 2, page 2.

CHAPTER 3: RESULTS AND CONCLUSIONS

Sample Problems

Each of the problems examined were first solved as constrained NLP's using the ellipsoid algorithm, EA3. Then, using the calculations demonstrated for the example problem in chapter 1, a penalty multiplier for each was determined. The problem was then solved as an unconstrained exact penalty function. The problems examined were:

Table I: Problem Summaries

PROBLEM	n= # VARIABLES	m= # CONSTRAINTS	REF
Example	2	4	N/A
Colville 1	5	15	[3]
BBZ 3	16	13	[2]
Powell 19	2	20	[4]
Fukushima	5	11	[8]

For the example problem, the strictly feasible point used was $\mathbf{z}=(1,1)$. Two lower bounds on the optimal objective value were calculated for use in determining a penalty multiplier. The first bound was based on the unconstrained minimization of the objective function (initial iteration of the gradient method for solving the Lagrangian Dual Problem). The second was based on knowledge of the optimal solution. Two corresponding penalty multipliers were then calculated. The same procedure was followed to calculate penalty multiplier values for the remainder of the sample problems. For each of

the other four sample problems, the procedure is only complicated by the larger number of variables in each problem and the larger number of constraint equations. The calculations for **z** and the lower bound on the objective function were simplified through the use of a Fortran program by Mike Kupferschmid, AM46:FEASTEST. This program examines a given point, **x**, and determines if any of the constraint equations in the FCN routine are violated. The program also lists the constraints in order, from closest to being violated to farthest. From the screen output, the **x** being investigated is clearly strictly feasible or not. Also the α value can be picked right from the rank order of constraints. Results for these preliminary calculations are shown here:

Table II: Penalty Multiplier Calculations

PROBLEM	Z=STRICT FEAS PT	F'=LOW BND	C=PEN MULT
Example	[1]	0 and 8	19 and 9
Colville 1	[.1 .1 .2 .6 .5]	-61.448	443.56
BBZ3	[.1 .6 0 -1.5 -3.7 -2.5 -.8 .7 5 0 .6 .2]	30.62331	41
Powell 19	[-.5 -.5]	-1.9999987	7
Fukushima	[0 1 2 -1 10]	-353.0495	11

Optimal Value Results

The sample problems were solved using the ellipsoid algorithm on each of the two forms of the problem (constrained and penalty function). The best **feasible** solution point was

then found for each solution process and these results were then tabulated for comparison:

Table III: Optimal Value Results

PROBLEM	CONSTRAINED SOLUTION	PENALTY FCN SOLUTION
Example	8.000000000000009	**8.000000000000000**
Colville 1	**-32.34867896572270**	**-32.34867896572270**
BBZ 3	**30.62330955021475**	30.62334552248174
Powell 19	**-1.414231127874633**	**-1.414231127874633**
Fukushima	**-43.99999999999998**	**-43.99999999999998**

The best solution is shown in bold for each of the problems. In the case of the simple example problem, the penalty function formulation solved the NLP to the exact solution of 8. Solution of the constrained problem yielded an optimal value within 10^{-15} of the exact solution. For the problems Colville 1, Powell 19, and Fukushima, both formulations yielded the same result for the optimal values. In the case of BBZ 3, the solution of the exact penalty function formulated problem yielded a solution within 10^{-5} of the solution to the constrained problem. These results indicate that for convex problems, the solution of the reformulated NLP achieves basically equal results with the constrained solution.

Time to Solution Results

The other part of the comparison of the methods of formulation is to see if the reformulated problem can be

solved faster. Intuitively, the solution to an unconstrained problem would be easier or faster to achieve as compared to a constrained problem. After each problem was solved in each of the two formulations, the largest time difference between iterations was calculated, and the results are presented here:

Table IV: Solution Time Results

	LARGEST TIME DIFFERENCE (% FASTER)	
PROBLEM	CONSTRAINED PROBLEM	PENALTY FCN SOLUTION
Example	14.6 %	
Colville 1	23.9 %	
BBZ 3	54.2 %	
Powell 19	14.6 %	
Fukushima		6.4 %

These results show that in four of the five problems, the constrained formulation converged to a solution faster than the penalty function formulation. Only the Fukushima problem was solved faster in the unconstrained formulation. The error versus effort curves in appendix 2 support these results. The plots for Colville 1 (pg 2-3), BBZ 3 (pg 2-4), and Powell 19 (pg 2-5) all show steeper, hence faster, convergence for the constrained formulation. Only the Fukushima plot (pg 2-6) shows the unconstrained penalty function formulation converging at a faster rate.

The solution time results run counter to the intuitive expectations as to which formulation of the problem should be faster to solve. An explanation for these results comes from the manner in which the ellipsoid algorithm solves a problem. A standard NLP has "m" constraint equations with the objective

function labelled as the "m+1"st equation. Recall, the ellipsoid algorithm evaluates the constraints until one is found which is violated. The corresponding gradient for that constraint is then evaluated. These function and gradient evaluations are the timed segments in the effort portion of the plots. The number of overall function evaluations can, therefore, vary on each iteration:

1. **Best Case:** the first constraint examined is violated. The result is that only a single function and its gradient are evaluated. Total function evaluations: 2.
2. **Worst Case:** all of the constraints are satisfied (the current point is feasible). So, all m constraints are evaluated. The objective function and its gradient are evaluated. Total function evaluations: (m+1)+1= m+2.
3. **All Others:** an intermediate constraint is violated, so greater than two evaluations but less than m+2 evaluations are needed.

The exact penalty function formulation requires no function evaluations from FCN to solve the problem. The cutting hyperplane is determined based on the gradient of the only function present, the objective function. However, since the objective function includes a sum of all of the constraints (the p(**x**,c) penalty function), the gradient of all "m" constraints plus the original objective function gradient are calculated on each iterate. The result is "m+1" function evaluations on each iterate. No variations occur from iteration to iteration. Graphically, the number of evaluations per iteration can be shown as:

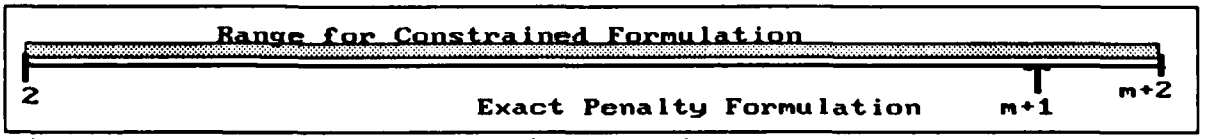


Figure 2: Function Evaluation Requirements

Conclusions

The exact penalty function formulation provides an alternative means to solve convex nonlinear programming problems. The theorem has been proved (appendix 1) to always converge to a solution for convex problems, achieving the same optimal point as when solving the standard constrained NLP, when analytic solutions are possible. Differences in the solution can occur due to numerical methods in the solution algorithm, as seen in the optimal solution results for the problems examined.

Zangwill presents a method to calculate a sufficient penalty multiplier to guarantee convergence to the same optimal point. The theorem also guarantees convergence for any multiplier greater than the one calculated. However, this penalty multiplier is not necessarily the minimum multiplier which will guarantee convergence. For the simple example problem worked throughout this project, two penalty multipliers were calculated (9 and 19). The error versus effort plot showing these two convergence trajectories is at page 2 of appendix 2. The multiplier 9 was based on the true objective value of 8 for the problem. However, lower penalty multipliers will also cause the solution to converge until a multiplier of 3 is chosen. The plot for convergence using a penalty multiplier of 3 is also shown on the same error versus effort plot. The plot shows that the solution diverges from the true optimal value, while the plots for multipliers 9 and 19 basically follow the same path.

In four out of five test problems, the reformulated unconstrained problem was solved at a slower rate than the original constrained problem. Only in the Fukushima problem was faster convergence achieved by the penalty function formulation. As shown in figure 2, the penalty function

formulation requires that m+1 functions be evaluated on each iteration. Meanwhile, each iteration of the constrained problem requires evaluations in the range: [2, m+2]. Therefore, for the reformulated problem to converge faster, the ellipsoid algorithm must demonstrate worst case behavior (every iterate, $\mathbf{x}^k$, is a feasible point). This worst case behavior is demonstrated in the Fukushima problem where virtually every iterate is feasible. In this case, the ellipsoid algorithm solves the reformulated unconstrained problem faster. However, this type of behavior in the solution process is more the exception rather than the norm.

REFERENCES

[1] M.S. Bazaraa and C.M. Shetty, Nonlinear Programming Theory and Algorithms, John Wiley, New York, 1979.

[2] A. Ben-Israel, A. Ben-Tal and S. Zlobec, Optimality in Nonlinear Programming, John Wiley, New York, 1981.

[3] A.R. Colville, **A Comparative Study of Nonlinear Programming Codes**, IBM New York Scientific Center Report 320-2949, International Business Machines Corporation, New York, 1968.

[4] J.G. Ecker, **NGC:Catalog**, MTS Filename, Rensselaer Polytechnic Institute, Troy, New York, 1990.

[5] J.G. Ecker and M. Kupferschmid, **A Computational Comparison of the Ellipsoid Algorithm with Several Nonlinear Programming Algorithms**, SIAM J. Control and Optimization, vol. 23, no. 5, September, 1985.

[6] J.G. Ecker and M. Kupferschmid, Introduction to Operations Research, John Wiley, New York, 1988.

[7] R. Fletcher, Practical Methods of Optimization, 2d ed., John Wiley, New York, 1987.

[8] M. Fukushima, **An Outer Approximation Algorithm for Solving General Convex Programs**, Operations Research, vol. 31, 1983.

[9] M. Kupferschmid and J.G. Ecker, **EA3: A Practical Implementation of the Ellipsoid Algorithm for Nonlinear Programming**, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York, 1984.

[10] D.G. Luenberger, Linear and Nonlinear Programming, 2d ed, Addison-Wesley, Reading, Massachusetts, 1989.

[11] W. Zangwill, **Nonlinear Programming via Penalty Functions**, Management Science, vol. 13, no. 5, January, 1967.

APPENDIX 1: PROOF OF ZANGWILL THEOREM

In [11], Zangwill states and proves his penalty function theorem for the concave nonlinear programming problem. Restated, the theorem for the convex nonlinear problem is as follows:

THEOREM: let $\mathbf{x}^*$ be the optimal point for the convex nonlinear programming problem:

$$\text{minimize} \quad f(\mathbf{x})$$

$$\text{s.t.} \quad g_i(\mathbf{x}) \leq 0 \ , \ i=1,2,\ldots,m$$

where $\mathbf{x} \in \mathbf{R}^n$ and f and the g_i's are convex

and, let $S^\circ=\{\mathbf{x} | g_i(\mathbf{x})<0, \ i=1,2,\ldots,m\}$ ≐ interior of the feasible set

and, assume: S° is nonempty, and $S^\circ=\text{int}(S)$

then, the convex NLP can be solved by a single unconstrained minimization

PROOF:

1. define: $p(x,c)=f(x)+c\sum_{i=1}^{m} \max[g_i(x),0]$

   then, minimizing $p(\mathbf{x},c)$ yields unconstrained NLP

2. find a $\bar{c}$, so that $\exists \ \min[p(x,\bar{c})]$

   ie: $p(x^*,\bar{c})=\min_x[p(x,\bar{c})]$ *where* x^* ≐ *optimal point*

   and:

3. by assumption, S° is nonempty so let: $\mathbf{z} \in S^\circ$,

   s.t. $g_i(\mathbf{z})<0$

   define: $\alpha=\max_i[g_i(z)]<0$

4. let $\mathbf{x}^*$ be the optimal point, then define:

   $\beta=f(x^*)-f(z)$ and $\bar{c}=\frac{\beta-1}{\alpha}$

5. now, given any point **w** , which is infeasible, finding a point **v** , feasible, where: $p(v,\bar{c}) < p(w,\bar{c})$, is sufficient to prove the theorem.

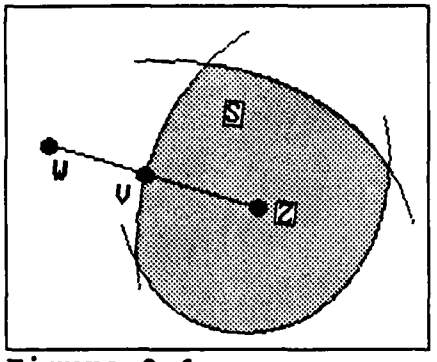


Figure 3.1

ie: $\min p(x,\bar{c})$ *on* $S - p(x^*,\bar{c}) - f(x^*)$

-now, for any **w** infeasible, show that there exists a **v** feasible, s.t.: $p(v,\bar{c}) < p(w,\bar{c})$

-this implies that the minimizing point for $p(x,\bar{c})$ over $\mathbf{R}^n$ must be in the feasible set.

6. define **v** as follows (see figure 1.1): $z \in S^\circ \quad w \notin S$

which implies that $\exists v \in \delta(S)$, and **v** lies on the line segment joining **z** and **w**

now: $v \in \delta(S)$ implies $\exists$ *set* $A - [i \mid g_i(v) - 0]$

and $\forall\, i \notin A,\ g_i(v) < 0$

7. define a function: $t(x) - f(x) + \bar{c} \sum_{i \in A} g_i(x)$, convex

so, since: $g_i(v) - 0,\ \forall i \in A \rightarrow t(v) - f(v) + \bar{c} \sum_{i \in A} g_i(v) - p(v,\bar{c}) - f(v)$

8. at **w**: $g_i(w) \geq 0,\ \forall i \in A$ (because **w** is infeasible)

so: $\sum_{i \in A} g_i(w) - \sum_{i \in A} \max[g_i(w), 0] \leq \sum_{i-1}^{m} \max[g_i(w), 0]$

9. $\rightarrow t(w) - f(w) + \bar{c} \sum_{i \in A} g_i(w) \leq f(w) + \bar{c} \sum_{i-1}^{m} \max[g_i(x), 0]$

or: $t(w) \leq p(w,\bar{c})$

* The proof began with premise of showing that $p(v,\bar{c}) < p(w,\bar{c})$

but: $t(v) = p(v,\bar{c}) \Rightarrow t(v) < p(w,\bar{c}) \geq t(w)$

therefore, to complete the proof, it is sufficient to show that: $t(\mathbf{v}) < t(\mathbf{w})$

10. first show $t(\mathbf{z}) < t(\mathbf{w})$:

$$t(z) = f(z) + \bar{c} \sum_{i \in A} g_i(z) = f(z) + \frac{\beta - 1}{\alpha} \sum_{i \in A} g_i(z)$$

$$t(z) \leq f(z) + \frac{\beta - 1}{\alpha} \max_i [g_i(z)] \text{ , because } g_i(\mathbf{z}) < 0 \text{ for all i}$$

$$t(z) \leq f(z) + \frac{f(x^*) - f(z) - 1}{\max_i [g_i(z)]} \max_i [g_i(z)]$$

$$t(z) \leq f(x^*) - 1$$

$$t(z) < f(v) \text{ , since } \mathbf{v} \text{ is feasible, } \mathbf{z} \text{ is strictly feasible}$$

$$\therefore t(z) < t(v) \text{ , since } t(\mathbf{v}) = f(\mathbf{v})$$

11. now show $t(\mathbf{v}) < t(\mathbf{w})$:

rewrite $\mathbf{v}$ as: $v = \lambda z + (1-\lambda) w$, $\lambda \in (0,1)$, and $t(\mathbf{x})$ convex

$$\Rightarrow t(v) \leq \lambda t(z) + (1-\lambda)\, t(w)$$

$$t(v) < \lambda t(v) + (1-\lambda)\, t(w) \text{ , since } t(\mathbf{z}) < t(\mathbf{v})$$

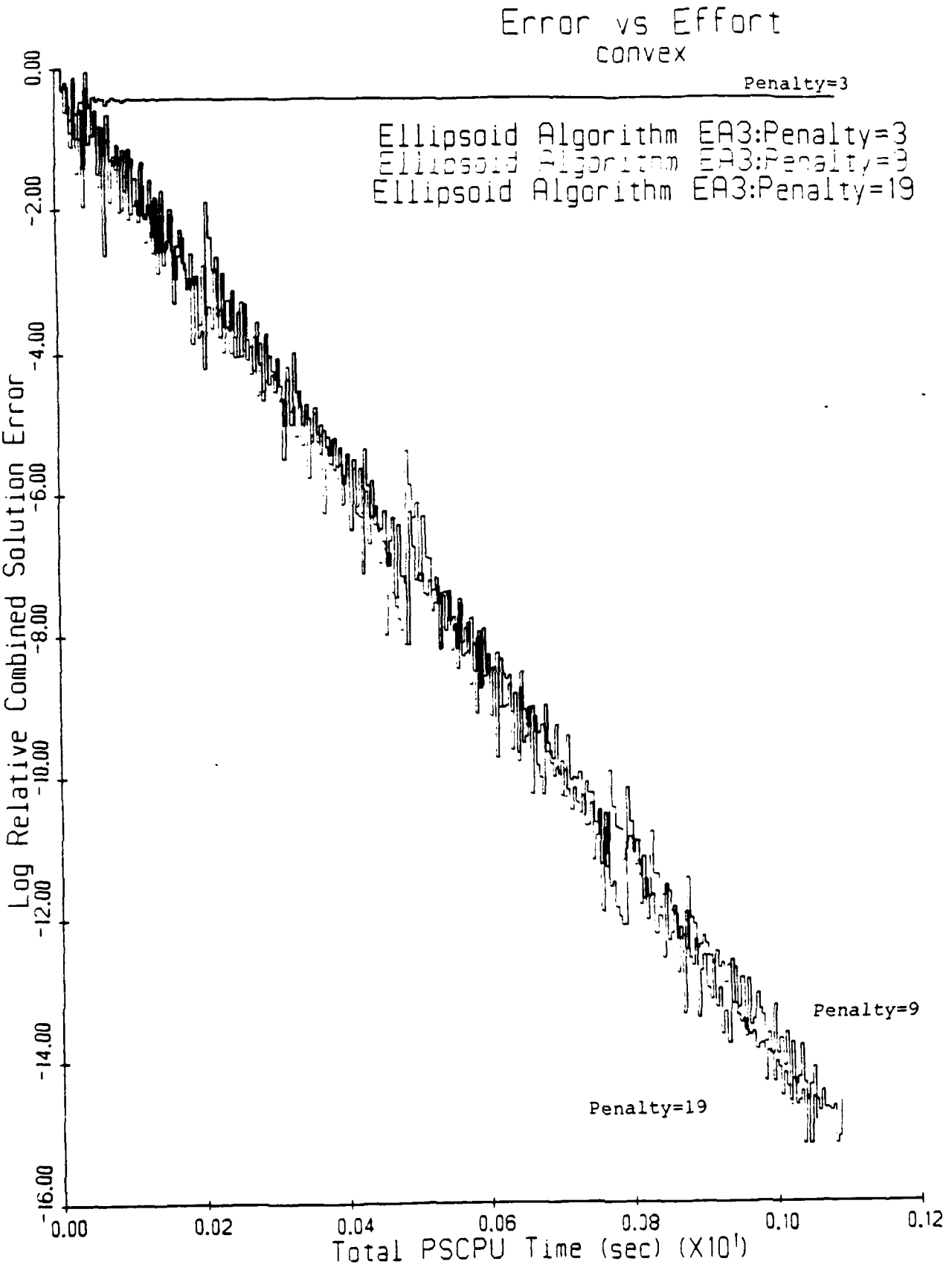
$$(1-\lambda)\, t(v) < (1-\lambda)\, t(w)$$

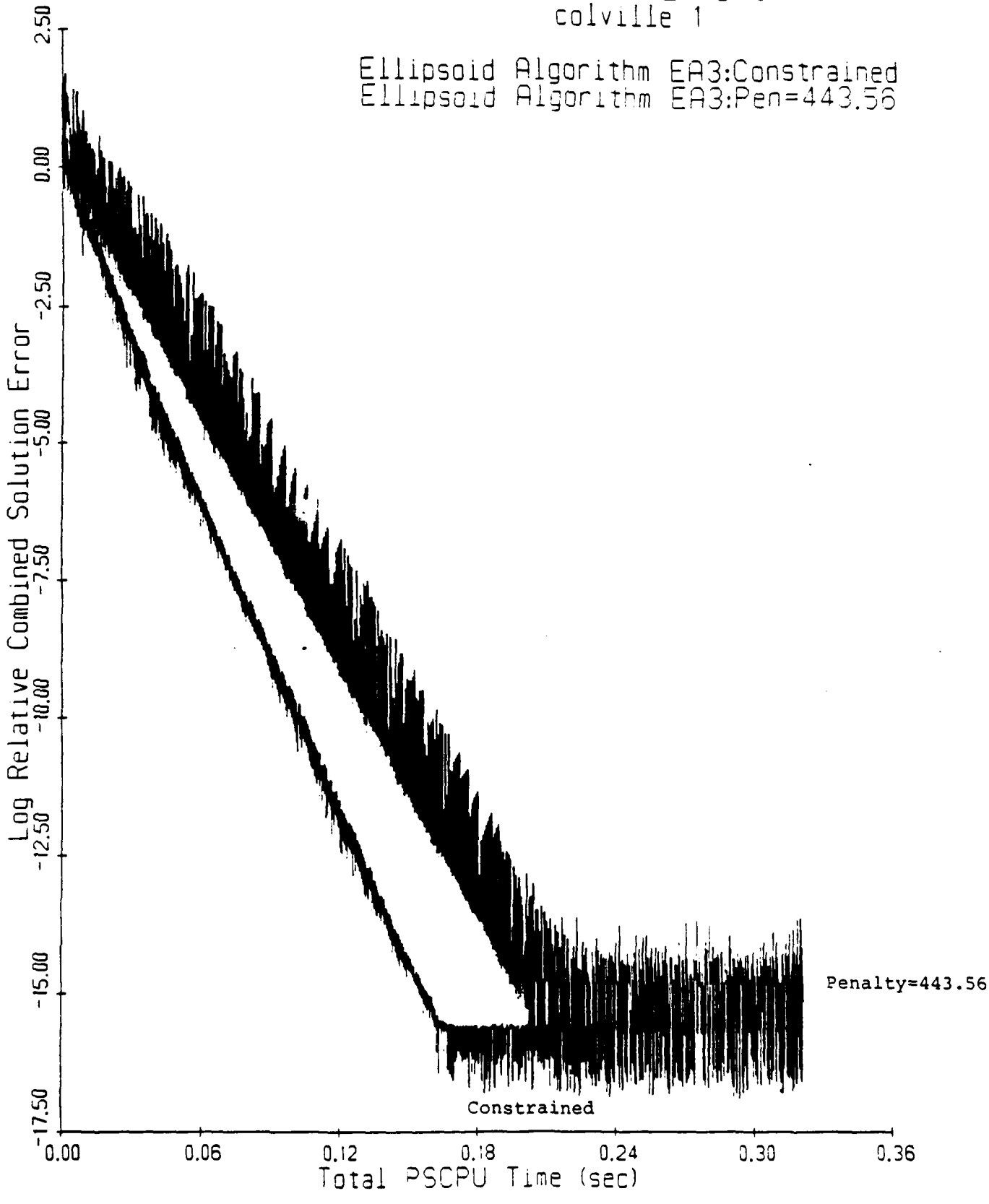
$$\therefore t(v) < t(w)$$

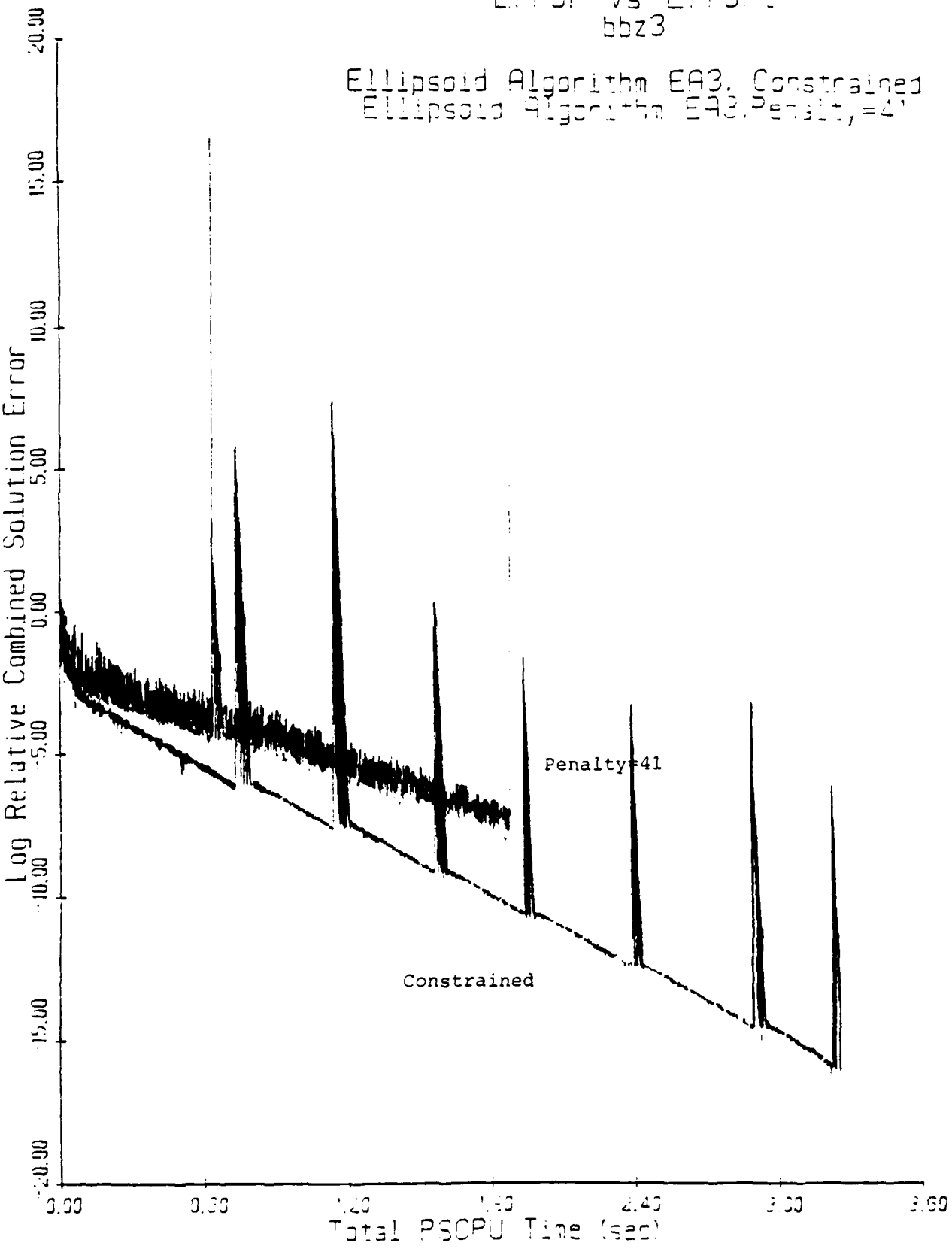
12. then $\mathbf{x}^*$ is the optimal point and the theorem holds $\forall\, c \geq \bar{c}$

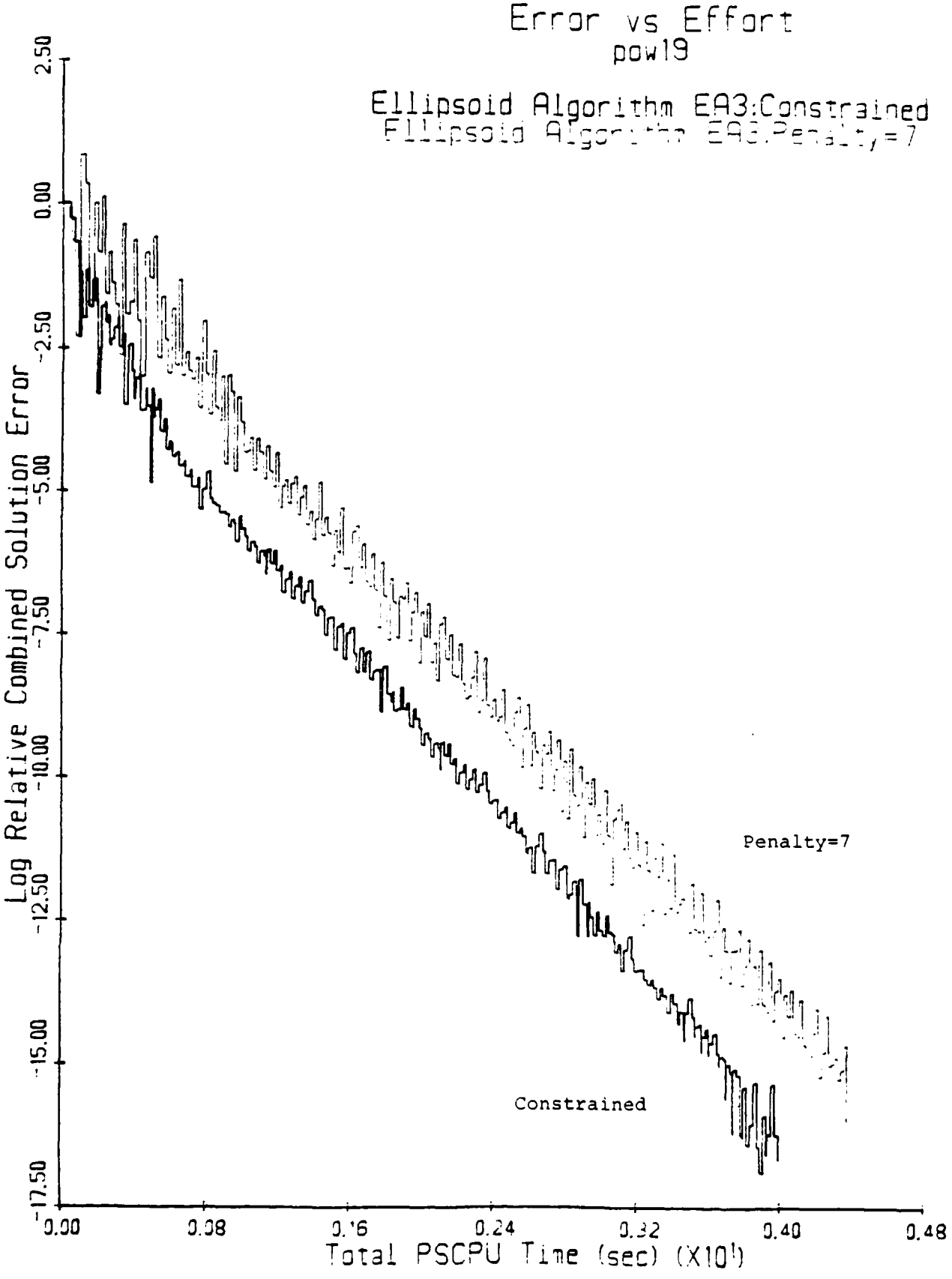
APPENDIX 2: ERROR VS EFFORT PLOTS

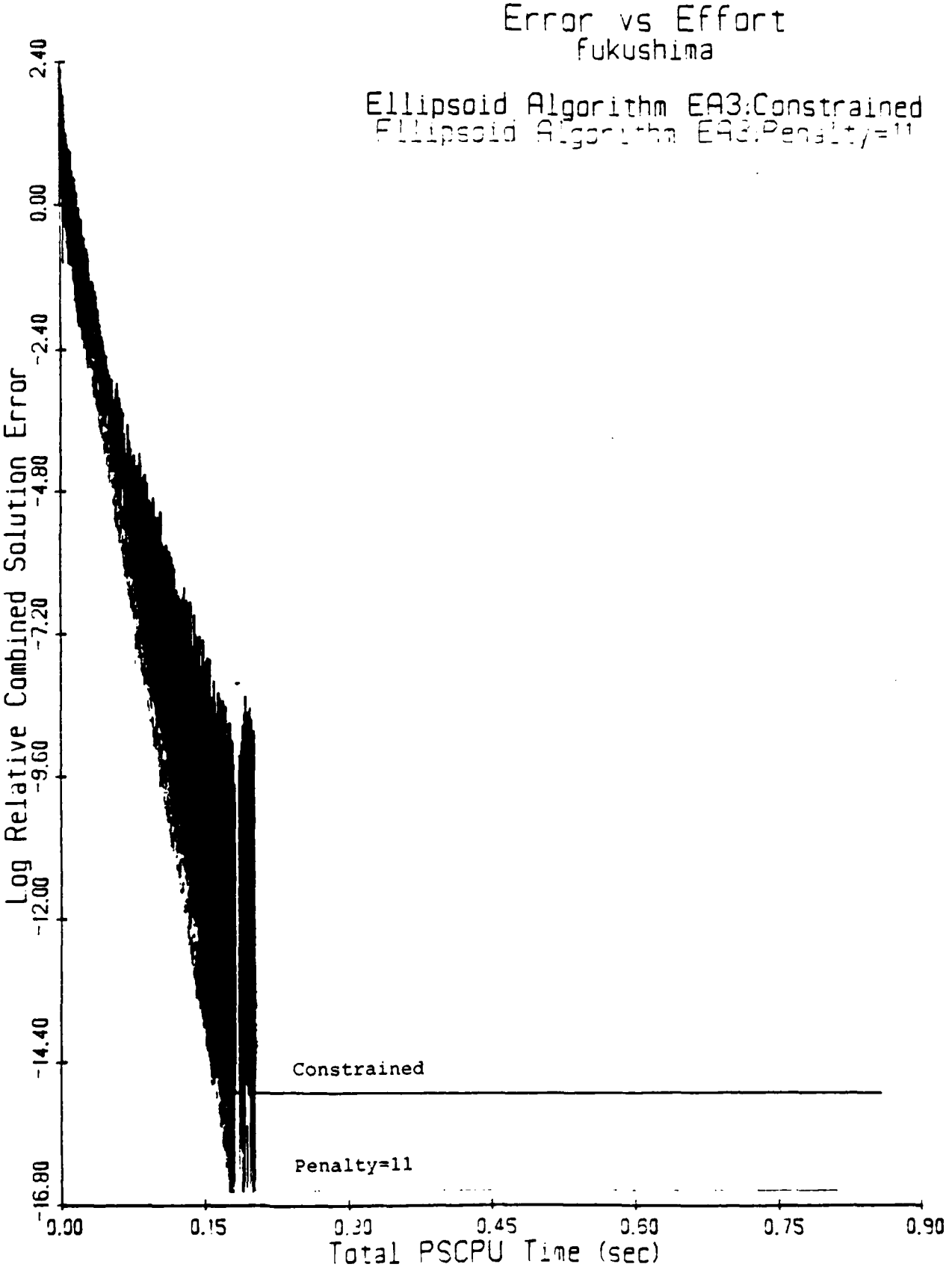
On the following pages are the error versus effort plots for the convex nonlinear programming problems examined during this study. They are included as follows:


Error vs Effort
convex
Penalty=3
Ellipsoid Algorithm EA3:Penalty=3
Ellipsoid Algorithm EA3:Penalty=9
Ellipsoid Algorithm EA3:Penalty=19
Penalty=9
Penalty=19
Log Relative Combined Solution Error
0.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Total PSCPU Time (sec) (X10¹)


Error vs Effort
colville 1
Ellipsoid Algorithm EA3:Constrained
Ellipsoid Algorithm EA3:Pen=443.56
Log Relative Combined Solution Error
2.50
0.00
-2.50
-5.00
-7.50
-10.00
-12.50
-15.00
-17.50
0.00
0.06
0.12
0.18
0.24
0.30
0.36
Total PSCPU Time (sec)
Constrained
Penalty=443.56


Error vs Effort
bbz3
Ellipsoid Algorithm EA3. Constrained
Ellipsoid Algorithm EA3.Penalty=41
Log Relative Combined Solution Error
20.00
15.00
10.00
5.00
0.00
-5.00
-10.00
-15.00
-20.00
0.00
0.80
1.20
1.60
2.40
3.00
3.60
Total PSCPU Time (sec)
Penalty=41
Constrained


Error vs Effort
pow19
Ellipsoid Algorithm EA3:Constrained
Ellipsoid Algorithm EA3:Penalty,=7
Log Relative Combined Solution Error
2.50
0.00
-2.50
-5.00
-7.50
-10.00
-12.50
-15.00
-17.50
0.00
0.08
0.16
0.24
0.32
0.40
0.48
Total PSCPU Time (sec) (X10¹)
Penalty=7
Constrained


Error vs Effort
fukushima
Ellipsoid Algorithm EA3:Constrained
Ellipsoid Algorithm EA3:Penalty=11
Log Relative Combined Solution Error
2.40
0.00
-2.40
-4.80
-7.20
-9.60
-12.00
-14.40
-16.80
0.00
0.15
0.30
0.45
0.60
0.75
0.90
Total PSCPU Time (sec)
Constrained
Penalty=11

APPENDIX 3: FUNCTION, GRADIENT ROUTINES

```
C
Code by Christopher Fowler                    RPI Troy, NY  12180
C
      DOUBLE PRECISION FUNCTION FCN(X,N,I)
C     This routine computes a function value for the example
C     problem used in the masters project.
C
C     variable  meaning
C     --------  -------
C     I         index of function whose value is wanted
C     N         number of variables (=2)
C     X         point at which the value is wanted
C
      REAL*8 X(N)
C
C-------------------------------------------------------------
Calculate the required constraint function value
C
      IF(I.EQ.1) FCN= X(1)-2.D0
      IF(I.EQ.2) FCN= X(2)-2.D0
      IF(I.EQ.3) FCN=-X(1)
      IF(I.EQ.4) FCN=-X(2)
C
Calculate the objective function value
C
      IF(I.EQ.5) FCN=(X(1)-4.D0)**2 + (X(2)-4.D0)**2
C
      RETURN
      END
C
Code by Christopher Fowler                    RPI Troy, NY  12180
C
      SUBROUTINE GRAD(X,N,I,G)
C     This routine computes a gradient vector for the example
C     problem used in the masters project.
C
C     variable  meaning
C     --------  -------
C     G         gradient vector returned
C     I         index of function whose gradient is wanted
C     J         index on the variables
C     N         number of variables (=2)
C     X         point at which the gradient is wanted
C
      REAL*8 X(N),G(N)
C
C-------------------------------------------------------------
```

```
C
C       start with the gradients of the constraints
C
        IF(I.NE.1) GOTO 2
      1 G(1)=1.D0
        G(2)=0.D0
        RETURN
C
      2 IF(I.NE.2) GOTO 3
        G(1)=0.D0
        G(2)=1.D0
        RETURN
C
      3 IF(I.NE.3) GOTO 4
        G(1)=-1.D0
        G(2)= 0.D0
        RETURN
C
      4 IF(I.NE.4) GOTO 5
        G(1)= 0.D0
        G(2)=-1.D0
        RETURN
C
C       now calculate the gradient of the objective fcn
C
      5 G(1)=2.D0*X(1)-8.D0
        G(2)=2.D0*X(2)-8.D0
C
        RETURN
        END
```

This is file CONVEX
It contains FCN and GRAD for an example nonlinear programming problem for the masters project.

APPENDIX 4: FCNX AND GRADX SUBROUTINES

I. Function Call (FCNX):

```
      BLOCK DATA
      COMMON /BOUNDS/ XHIN,XLIN,NIN,MIIN,MEIN
      REAL*8 XHIN(50)/50*Z8181818181818181/
      REAL*8 XLIN(50)/50*Z8181818181818181/
      INTEGER*4 NIN/Z81818181/,MIIN/0/,MEIN/0/
      END
C
Code by Michael Kupferschmid and Chris Fowler
C
      SUBROUTINE GET$IN
C     This routine gets the penalty mult at start of a run.
C
C     Variable  Meaning
C     --------  -------
C     CBAR      penalty multiplier
C     FREAD     system routine for free-format input
C     MEPF      number of original constraints
C     PROMPT    routine prompts for input from the terminal
C
C     send the penalty multiplier from common
      COMMON /EPF/ CBAR,MEPF
      REAL*8 CBAR
C
C     ------------------------------------------------------
C
C     prompt for and read number of original constraints
      CALL PROMPT('original constraints=',21)
      MEPF=0
      CALL FREAD('GUSER','INTEGER*4:',MEPF,&1)
C
C     prompt for and read the penalty multiplier
    1 CALL PROMPT('penalty multiplier=',20)
      CBAR=1.D+06
      CALL FREAD('GUSER','REAL*8:',CBAR,&2)
    2 RETURN
      END
C
Code by Mike Kupferschmid and Chris Fowler
C
      DOUBLE PRECISION FUNCTION FCNX(X,N,I)
C     This routine computes the function value for a
C     penalty function problem



C
```

```
C       Variable  Meaning
C       --------  -------
C       CBAR      penalty multiplier
C       DMAX1     Fortran fcn gives larger values REAL*8's
C       FCN       original function routine for problem
C       I         unused; index of the pen fcn objective(=1)
C       II        index of original fcn whose value is wanted
C       MEPF      number of original constraints
C       N         number of variables in problem
C       X         point at which fcn values are wanted
C
        REAL*8 X(N),FCN
C
C       receive the penalty mult from common
        COMMON /EPF/ CBAR,MEPF
        REAL*8 CBAR
C
C       ------------------------------------------------------
C
C       compute the value of the exact penalty function
C       objective function
        FCNX=FCN(X,N,MEPF+1)
C
C       constraints
        DO 1 II=1,MEPF
             FCNX=FCNX+CBAR*DMAX1(1.D0,FCN(X,N,II))
      1 CONTINUE
      RETURN
      END
```

II. Gradient Call (GRADX):

```
C
Code by Michael Kupferschmid and Chris Fowler
C
        SUBROUTINE GRADX(X,N,I,G)
C       This routine computes the gradient vector for the
C       penalty function problem
C
C       Variable  Meaning
C       --------  -------
C       CBAR      penalty multiplier
C       FCN       original function routine for the problem
C       G         gradient vector returned
C       GC        gradient vector of a constraint function
C       GRAD      original gradient subroutine
C       I         unused;index of pen fcn objective(=1)
C       II        index of orig fcn for which gradient needed
C       J         index on the variables
C       MEPF      number of original constraints
```

```
C       N          number of variables in problem
C       X          Point at which function values wanted
C
        REAL*8 X(N),G(N)
        REAL*8 GC(50),FCN
C
C       receive the penalty multiplier from common
        COMMON /EPF/ CBAR,MEPF
        REAL*8 CBAR
C
C       ---------------------------------------------------------
C
C       compute the gradient of the exact pen fcn objective
        CALL GRAD(X,N,MEPF+1,G)
C
C       constraints
        DO 1 II=1,MEPF
C            check whether the constraint is satisfied at X
             IF(FCN(X,N,II).LE.0.D0) GO TO 1
C
C            constraint is violated; add in CBAR*(natural
C            gradient)
             CALL GRAD(X,N,II,GC)
             DO 2 J=1,N
             G(J)=G(J)+CBAR*GC(J)
        2    CONTINUE
        1 CONTINUE
        RETURN
        END
```